(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 292 066 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
12.03.2003 Bulletin 2003/11

(51) Int Cl.[7]: **H04L 9/06**

(21) Application number: **02019688.7**

(22) Date of filing: **04.09.2002**

(84) Designated Contracting States:
AT BE BG CH CY CZ DE DK EE ES FI FR GB GR IE IT LI LU MC NL PT SE SK TR
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **08.09.2001 GB 0121793**

(71) Applicant: **Amphion Semiconductor Limited**
Belfast BT9 5BS (GB)

(72) Inventors:
- **Hu, Yi**
  Belfast (GB)
- **McLoone, Maire Patricia**
  Glenties, County Donegal (GB)

(74) Representative: **Wallace, Alan**
F. R. Kelly & Co.
9 University Street
Belfast BT7 1FY (GB)

(54) An apparatus for generating encryption or decryption keys

(57) The invention provides an apparatus for generating a plurality of sub-keys from a primary key comprising a plurality of data words. The apparatus comprises a shift register for storing the primary key; and a transformation apparatus arranged to perform one or more logical operations on respective data words from the shift register to produce a new data word. The arrangement is such that the new data word is loaded into the shift register, whereupon one of the data words stored in said shift register is shifted out of the shift register, the sub-keys being comprised of one or more of the output data words. The apparatus is particularly suitable for on-the-fly Rijndael decryption Round key calculation. In this context, the invention obviates the need to store the expanded key or to wait until the expanded key is generated from the cipher key before beginning decryption. This removes a latency of at least 10 clock cycles in the operation of the decryption apparatus.

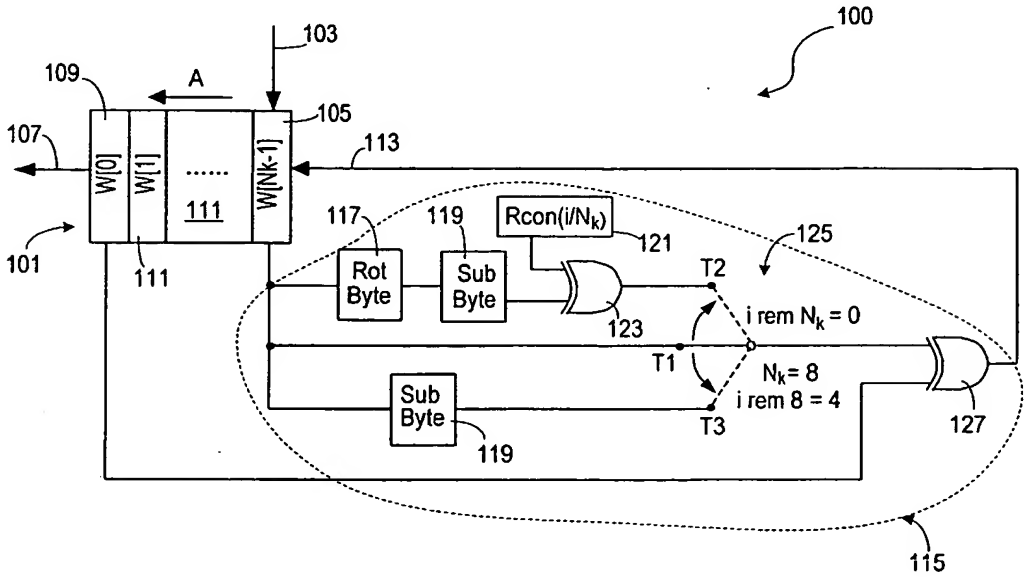

Fig. 9

EP 1 292 066 A1

Description

FIELD OF THE INVENTION

[0001] The present invention relates to the field of data encryption. The invention relates particularly to an apparatus for generating data encryption or decryption keys.

BACKGROUND TO THE INVENTION

[0002] Secure or private communication, particularly over a telephone network or a computer network, is dependent on the encryption, or enciphering, of the data to be transmitted. One type of data encryption, commonly known as private key encryption or symmetric key encryption, involves the use of a key, normally in the form of a pseudo-random number, or code, to encrypt data in accordance with a selected data encryption algorithm (DEA). To decipher the encrypted data, a receiver must know and use the same key in conjunction with the inverse of the selected encryption algorithm. Thus, anyone who receives or intercepts an encrypted message cannot decipher it without knowing the key.

[0003] Data encryption is used in a wide range of applications including IPSec Protocols, ATM Cell Encryption, Secure Socket Layer (SSL) protocol and Access Systems for Terrestrial Broadcast.

[0004] In September 1997 the National Institute of Standards and Technology (NIST) issued a request for candidates for a new Advanced Encryption Standard (AES) to replace the existing Data Encryption Standard (DES). A data encryption algorithm commonly known as the Rijndael Block Cipher was selected for the new AES.

[0005] As part of the Rijndael encryption process, the cipher key is expanded to produce an expanded key from which a number of sub-keys, or round keys, can be selected. Round keys are also required during decryption. The present invention concerns improvements in the generation of round keys for both encryption and decryption and relates particularly, but not exclusively, to the Rijndael cipher.

Summary of the Invention

[0006] A first aspect of the present invention provides an apparatus for generating a plurality of sub-keys from a primary key comprising a plurality of data words, the apparatus comprising: a shift register having a plurality of storage locations one for each data word of the primary key; and a transformation apparatus arranged to perform one or more logical operations on respective data words from at least two of said storage locations to produce a new data word, the arrangement being such that said new data word is loaded into a first of said storage locations, whereupon the data words stored in said shift register are shifted to a respective successive storage location and the data word in a final of said storage locations is output from said shift register, said sub-keys being comprised of one or more of said output data words.

[0007] The apparatus of the invention, when implemented in hardware, is relatively small in comparison to conventional solutions particularly since it avoids using multiplexers, or other switches, when selecting and distributing sub-keys. Further, the invention allows on-the-fly Rijndael decryption Round key calculation. This is advantageous as it obviates the need to store the expanded key or to wait until the expanded key is generated from the cipher key before beginning decryption. This removes a latency of at least 10 clock cycles in the operation of a data decryption apparatus.

[0008] Preferably, said new data word is loaded into said first storage location via a first switch, said switch being arranged to select which of said storage locations serves as said first storage location. More preferably, said at least one data word is provided to said transformation module from said shift register via a second switch, the second switch being arranged to select from which storage location said at least one data word is provided.

[0009] In the preferred embodiment, the transformation apparatus is arranged to perform transformations according to the Rijndael block cipher.

[0010] In one embodiment, the shift register is initialised with a primary key comprising a Rijndael cipher key and said transformation apparatus is arranged to perform said one or more logical operations on the respective data words stored in said first and said final storage locations.

[0011] In an alternative embodiment, the shift register is initialised with a primary key comprising a Rijndael inverse cipher key and said transformation apparatus is arranged to perform said one or more logical operations on the respective data words stored in said final storage location and the penultimate storage location.

[0012] A second aspect of the invention provides a method of generating a plurality of sub-keys from a primary key comprising a plurality of data words, method comprising: loading the primary key into a shift register having a plurality of storage locations one for each data word of the primary key; performing one or more logical operations on respective data words from at least two of said storage locations to produce a new data word; loading said new data word into a first of said storage locations, whereupon the data words stored in said shift register are shifted to a respective successive storage location and the data word in a final of said storage locations is output from said shift register, said

sub-keys being comprised of one or more of said output data words.

[0013] A third aspect of the invention provides a data encryption and/or decryption apparatus comprising the apparatus for generating a plurality of sub-keys according to the first aspect of the invention.

[0014] A fourth aspect of the invention comprises a computer program product comprising computer usable instructions for generating the apparatus of the first aspect of the invention.

[0015] An apparatus according to the first or third aspects of the invention may be implemented in a number of conventional ways, for example as an Application Specific Integrated Circuit (ASIC) or a Field Programmable Gate Array (FPGA). The implementation process may also be one of many conventional design methods including standard cell design or schematic entry/layout synthesis. Alternatively, the apparatus may be described, or defined, using a hardware description language (HDL) such as VHDL, Verilog HDL or a targeted netlist format (e.g. xnf, EDIF or the like) recorded in an electronic file, or computer useable file.

[0016] Thus, the invention further provides a computer program, or computer program product, comprising program instructions, or computer usable instructions, arranged to generate, in whole or in part, an apparatus according to the first or third aspects of the invention. The apparatus may be implemented as a set of suitable such computer programs. Typically, the computer program comprises computer usable statements or instructions written in a hardware description, or definition, language (HDL) such as VHDL, Verilog HDL or a targeted netlist format (e.g. xnf, EDIF or the like) and recorded in an electronic or computer usable file which, when synthesised on appropriate hardware synthesis tools, generates semiconductor chip data, such as mask definitions or other chip design information, for generating a semiconductor chip. The invention also provides said computer program stored on a computer useable medium. The invention further provides semiconductor chip data, stored on a computer usable medium, arranged to generate, in whole or in part, an apparatus according to the first or third aspects of the invention.

[0017] Other aspects of the invention will be apparent to those ordinarily skilled in the art upon review of the following description of specific embodiments and with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0018] Embodiments of the invention are now described by way of example and with reference to the accompanying drawings in which:

Figure 1a is a representation of data bytes arranged in a State rectangular array;

Figure 1b is a representation of a cipher key arranged in a rectangular array;

Figure 1c is a representation of an expanded key schedule;

Figure 2 is a schematic illustration of the Rijndael Block Cipher;

Figure 3 is a schematic illustration of a normal Rijndael Round;

Figure 4 is a schematic illustration of how round keys are required during Rijndael encryption;

Figure 4a is a schematic illustration of how round keys are required during Rijndael decryption;

Figure 5a is a schematic representation of an encryption apparatus for implementing the Rijndael cipher;

Figure 5b is a schematic representation of a decryption apparatus for implementing the Rijndael cipher

Figure 6 shows a flow chart for implementing the Rijndael key schedule for a 128-bit cipher key;

Figure 6a shows a flow chart for implementing the Rijndael key schedule for a 192-bit cipher key;

Figure 6b shows a flow chart for implementing the Rijndael key schedule for a 256-bit cipher key;

Figure 7 shows a composite flow chart for implementing the Rijndael key schedule for 128-bit, 192-bit or 256-bit cipher key;

Figure 8 shows a composite flow chart for implementing the Rijndael key schedule for 128-bit, 192-bit or 256-bit inverse cipher key;

Figure 9 shows, in general schematic view, an apparatus according to the invention for implementing Rijndael key expansion during encryption;

Figure 9a shows a specific embodiment of the apparatus of Figure 9 where $N_k = 4$;

Figure 9b shows an alternative embodiment of the apparatus of Figure 9 where $N_k = 4$, 6 or 8;

Figure 10 shows, in general schematic view, an apparatus according to the invention for implementing Rijndael key expansion during decryption using an inverse cipher key;

Figure 10a shows a specific embodiment of the apparatus of Figure 10 where $N_k = 4$;

Figure 10b shows a further embodiment of the apparatus of Figure 10 where $N_k = 4$, 6 or 8; and

Figure 11 shows values for use in a Look-Up Table (LUT) for implementing the Rijndael ByteSub transformation.

DETAILED DESCRIPTION OF THE DRAWINGS

[0019] The Rijndael algorithm is a private key, or symmetric key, DEA and is an iterated block cipher. The Rijndael algorithm (hereinafter "Rijndael") is defined in the publication "The Rijndael Block Cipher: AES proposal" by J. Daemen and V. Rijmen presented at the First AES Candidate Conference (AES1) of August 20-22, 1998, the contents of which publication are hereby incorporated herein by way of reference.

[0020] In accordance with many private key DEAs, including Rijndael, encryption is performed in multiple stages, commonly known as iterations, or rounds. Each round uses a respective sub-key, or round key, to perform its encryption operation. The round keys are derived from a primary key, or cipher key.

[0021] The data to be encrypted, sometimes known as plaintext, is divided into blocks for processing. Similarly, data to be decrypted is processed in blocks. With Rijndael, the data block length and cipher key length can be 128, 192 or 256 bits. The NIST requested that the AES must implement a symmetric block cipher with a block size of 128 bits, hence the variations of Rijndael which can operate on larger block sizes do not form part of the standard itself. Rijndael also has a variable number of rounds namely, 10, 12 and 14 when the cipher key lengths are 128, 192 and 256 bits respectively.

[0022] With reference to Figure 1a, the transformations performed during the Rijndael encryption operations consider a data block as a 4-column rectangular array, or State (generally indicated at 10 in Figure 1a), of 4-byte vectors, or words, 12. For example, a 128-bit plaintext (i.e. unencrypted) data block consists of 16 bytes, B_0, B_1, B_2, B_3, B_4... B_{14}, B_{15}. Hence, in the State 10, B_0 becomes $P_{0,0}$, B_1 becomes $P_{1,0}$, B_2 becomes $P_{2,0}$ ... B_4 becomes $P_{0,1}$ and so on.

[0023] Figure 1a shows the state 10 for the standards compliant 128-bit data block length. For data block lengths of 192-bits or 256-bits, the state 10 comprises 6 and 8 columns of 4-byte vectors respectively. It will be understood that the term 'word' as used herein refers to a basic unit or block of data and is not intended to imply any particular size.

[0024] With reference to Figure 1b, the cipher key is also considered to be a multi-column rectangular array 14 of 4-byte vectors, or words, 16, the number of columns, N_k, depending on the cipher key length. Thus, for cipher key lengths of 128-bits, 192-bits and 256 bits, the key block length N_k is 4, 6 and 8 respectively. In Figure 1b, the vectors 16 headed by bytes $K_{0,4}$ and $K_{0,5}$ are present when the cipher key length is 192-bits or 256-bits, while the vectors 16 headed by bytes $K_{0,6}$ and $K_{0,7}$ are only present when the cipher key length is 256-bits.

[0025] Referring now to Figure 2, there is shown, generally indicated at 20, a schematic representation of Rijndael. The algorithm design consists of an initial data/key addition operation 22, in which a plaintext data block is added to the cipher key, followed by nine, eleven or thirteen rounds 24 when the key length is 128-bits, 192-bits or 256-bits respectively and a final round 26, which is a variation of the typical round 24. There is also a key schedule operation 28 for expanding the cipher key in order to produce a respective different round key for each round 24, 26.

[0026] Figure 3 illustrates the typical Rijndael round 24. The round 24 comprises a ByteSub transformation 30, a ShiftRow transformation 32, a MixColumn transformation 34 and a Round Key Addition 36. The ByteSub transformation 30, which is also known as the *s-box* of the Rijndael algorithm, operates on each byte in the State 10 independently.

[0027] The transformations and other operations (including logical operations) involved in the normal round 24 and the final round 26 are defined in the Rijndael specification referred to above and may be implemented in a number of conventional ways.

[0028] The Rijndael key schedule 28 consists of two parts: Key Expansion and Round Key Selection. Key Expansion involves expanding the cipher key into an expanded key, namely a linear array 15 (Fig. 1c) of 4-byte vectors or words 17, the length of the array 15 being determined by the data block length, N_b, (in bytes) multiplied by the number of rounds, N_r plus 1, i.e. array length = $N_b * (N_r + 1)$. In standards-compliant Rijndael, the data block length is four words,

N_b = 4. When the key block length, N_k = 4, 6 and 8, the number of rounds is 10, 12 and 14 respectively. Hence the lengths of the expanded key are as shown in Table 1 below.

Table 1.

Length of Expanded Key for Varying Key Sizes			
Data Block Length, N_b	4	4	4
Key Block Length, N_k	4	6	8
Number of Rounds, N_r	10	12	14
Expanded Key Length	*44*	*52*	*60*

[0029] The first N_k words of the expanded key comprise the cipher key. When N_k = 4 or 6, each subsequent word, W[i], is found by XORing the previous word, W[*i*-1], with the word N_k positions earlier, W[i-N_k]. For words 17 in positions which are a multiple of N_k, a transformation is applied to W[i-1] before it is XORed. This transformation involves a cyclic shift of the bytes in the word 17. Each byte is passed through the Rijndael s-box 30 and the resulting word is XORed with a round constant stipulated by Rijndael (see *Rcon(i)* function described below). However, when N_k = 8, an additional transformation is applied: for words 17 in positions which are a multiple of ((N_k*i)+ 4), each byte of the word, W[i-1], is passed through the Rijndael s-box 30.

[0030] The round keys are selected from the expanded key 15. In a design with N_r rounds, N_r +1 round keys are required.

For example a 10-round design requires 11 round keys. Round key 0 comprises words W[0] to W[3] of the expanded key 15 (i.e. round key 0 corresponds with the cipher key itself) and is utilised in the initial data/key addition 22, round key 1 comprises W[4] to W[7] and is used in round 0, round key 2 comprises W[8] to W[11] and is used in round 1 and so on. Finally, round key 10 is used in the final round 26.

[0031] The decryption process in Rijndael is effectively the inverse of its encryption process. Decryption comprises an inverse of the final round 26, inverses of the rounds 24, followed by the initial data/key addition 22. The encryption process is described in the Rijndael specification and may be implemented in a number of conventional ways.

[0032] The same cipher key is used for decryption as was used to encrypt the data. Therefore, during decryption, the key schedule 28 does not change. However, the round keys constructed for encryption (i.e. during the key expansion described above) are now used in reverse order. For example, in a 10-round design, round key 0 is still utilized in the initial data/key addition 22 and round key 10 in the inverse of the final round 26. However, round key 1 is now used in round 8, round key 2 in round 7 and so on. Figures 4 and 4a illustrate how the round keys, denoted as *Rnd Key* in Figures 4 and 4a, are required by each round 24, 26 during encryption and decryption respectively.

[0033] Normally, all of the round keys are generated from the cipher key before decryption can begin (since the round keys are required in reverse order during decryption). This normally introduces a delay into the decryption process since the decryption apparatus has to wait a number of clock cycles (10 clock cycles in the 10-round example above) before the relevant round keys are available. Further, the round keys need to be stored until they are needed - this is conveniently done by using data registers. Alternatively, the round keys can be precomputed and stored in memory until they are required by the decryption apparatus.

[0034] A further alternative is to calculate the round keys for decryption by using the last N_k words created during key expansion in the encryption process as the cipher key for decryption - the last N_k words are known as the inverse cipher key. By expanding the inverse cipher key, the round keys can be created as they are required by the inverse rounds during decryption. Since encryption is always performed prior to decryption, the inverse cipher key is readily available as it is produced during key expansion for encryption. Thus, there is no need to wait until all the round keys are available before beginning decryption, and there is no need to provide means for storing the round keys as described above.

[0035] A number of different architectures can be considered when designing an apparatus or circuit for implementing encryption algorithms. These include Iterative Looping (IL), where only one data processing module is used to implement all of the rounds. Hence for an n-round algorithm, n iterations of that round are carried out to perform an encryption, data being passed through the single instance of data processing module n times. Loop Unrolling (LU) involves the unrolling of multiple rounds. Pipelining (P) is achieved by replicating the round i.e. devising one data processing module for implementing the round and using multiple instances of the data processing module to implement successive rounds. In such an architecture, data registers are placed between each data processing module to control the flow of data. A pipelined architecture generally provides the highest throughput. Sub-Pipelining (SP) can be carried out on a partially pipelined design when the round is complex. This decreases the pipeline's delay between stages but increases the number of clock cycles required to perform an encryption.

[0036] The present invention relates to an apparatus for generating round keys for use in a data encryption and/or data decryption apparatus. The invention is not limited to use with any particular types of architecture for the overall encryption/decryption apparatus. However, the embodiments of the invention described herein relate particularly to the case where each encryption or decryption round is performed in four clock cycles (where $N_b = 4$ and each cycle processes 32-bits at a time), irrespective of whether the overall encryption/decryption apparatus is iterative or pipelined. It will be understood that the invention applies equally where $N_b = 6$ or 8, in which cases the rounds are performed in 6 and 8 cycles respectively and complete round keys are produced every 6 and 8 clock cycles respectively.

[0037] Referring now to Figure 5a, there is shown, for illustrative purposes only, an apparatus 40 for encrypting blocks of data. The apparatus 40 is arranged to receive a plaintext input data block (shown as "plaintext" in Figure 5a) and a cipher key (shown as "key" in Figure 5a) and to produce, after a number of encryption rounds, an encrypted data block (shown as "ciphertext" in Figure 5a).

[0038] The apparatus 40 comprises a data/key addition module 48 for performing the data/key addition operation 22 (Figure 2). The Data/Key Addition module 48 conveniently comprises an XOR component (not shown) arranged to perform a bitwise XOR operation of each byte B_i of the State 10 comprising the input plaintext, with a respective byte K_i of the cipher key.

[0039] The apparatus 40 also includes a data processing module in the form of a round module 44 for implementing the encryption rounds 24. In the illustrated example, the data block length N_b is assumed to be 128-bits. The data/key addition module 48 provides, via a 2-to-1 switch or multiplexer 60, the result of the data/key addition operation to the round module 44. In the present example, the result of the data/key addition operation comprises 128-bits of data and control circuitry 58 is arranged to control the switch 60 to supply this data the round module 44. The control circuitry 58 then controls the switch 60 to implement a feedback loop from the output of the round module 44. In the present example, the round module 44 is arranged to perform encryption operations on one quarter of the received data, in this case 32-bits, in each clock cycle. Thus, the round module 44 performs one round transform every four clock cycles, the first four clock cycles producing the result of round 0, the next four clock cycles producing the result of round 1, and so on. Once all of the required encryption rounds are completed, the encrypted data is provided to a final round module 46 which implements the Rijndael final round to produce the output ciphertext.

[0040] Figure 5b shows a data decryption apparatus 40' of generally similar iterative design as the encryption apparatus 40. The decryption apparatus 40' is arranged to receive a ciphertext input data block and an inverse cipher key and to produce, after a number of decryption rounds, a decrypted data block (plaintext). In the decryption apparatus 40' the respective positions of the data/key addition module 48' and the inverse final round module 46' are interchanged and the round module 44' is arranged to perform the inverse of the encryption round.

[0041] In each case, the encryption apparatus 40 and decryption apparatus 40' each include a key schedule module 50, 50' arranged to implement the key schedule 28. The key schedule modules 50, 50' are arranged to receive the cipher key and the inverse cipher key, respectively, and to generate the round keys, or sub-keys, as they are required by the respective round modules 44, 44', 46, 46'. In the present example, the key schedulers 50, 50 produce a round key over four consecutive clock cycles and thus the production of round keys is synchronised with the four clock cycle round transformation implemented by the round modules 44, 44'. The respective control circuitry 58, 58' receives the round keys from the key schedule modules 50, 50' and distributes them to the round modules 44, 44', 46, 46' as required. The final round 46 and the inverse final round 46' may be arranged to operate on 128-bits at a time (i.e. to perform its round transformation in one clock cycle) or on 32-bits at a time (i.e. to perform its round transformation in four clock cycles) as desired and the control circuitry 58, 58' may be arranged to provide the respective round key accordingly.

[0042] The present invention concerns in particular the implementation of the key schedulers 50, 50' as is described in more detail hereinafter.

[0043] In Figure 6, there is shown a flow chart illustrating the key expansion part (operations 905 to 945) and the round key selection part (operations 955 to 970) included in the key schedule 28. The flow chart of Figure 6 relates to the case where the key block length $N_k = 4$, the data block length $N_b = 4$ and the number of rounds $N_r = 10$. Alternative flow charts are given in Figures 6a and 6b for the case where the key lengths are 192 bits and 256 bits respectively. Figure 7 shows a composite flow chart for implementing the Rijndael key schedule when the key length is 128-bits, 192-bits or 256-bits. The flow charts of Figures 6a, 6b and 7 will be readily understood by persons skilled in the art by analogy with the following description of Figure 6.

[0044] Referring now to Figure 6 (numerals in parentheses() referring to the drawing labels), the input to the key schedule module 50 is the cipher key which is assigned to the first four words W[0] to W[3] of the expanded key (905). A counter *i* (which represents the position of a word within the expanded key) is set to four (910). The word W[*i*-1] (which initially is W[3]) is assigned to a 4-byte word *Temp* (915). A remainder function *rem* is performed on the counter *i* to determine if its current value is a multiple of N_k, which in the present example is equal to 4 (920). If the result of the *rem* function is not zero i.e. if the counter value is not exactly divisible by 4, then the word W[*i*-4] is XORed with the word currently assigned to *Temp* to produce the next word W[*i*] (950). For example, when $i = 5$, W[5] is produced

by XORing W[1] with W[4].

[0045] The value of counter *i* is then tested to check if all the words of the expanded key have been produced - 44 words are required in the present example (945). If *i* is less than 44 i.e. the expanded key is not complete, then counter *i* is incremented (946) and control returns to step 915.

[0046] If the result of the *rem* function is zero (920), this indicates that the word currently assigned to *Temp* is in a position that is a multiple of N_k and so requires to undergo a transformation. A function *RotByte* is performed on the word assigned to *Temp*, the result being assigned to a 4-byte word *R* (925). The *RotByte* function involves a cyclical shift to the left of the bytes in a 4-byte word. For example, an input of (B_0, B_1, B_2, B_3) will produce the output (B_1, B_2, B_3, B_0).

[0047] A function *SubByte* is then performed on R (930), the result being assigned to a 4-byte word S. *SubByte* operates on a 4-byte word and involves subjecting each byte to the ByteSub transformation 30 described above.

[0048] The resulting word S is XORed with the result of a function *Rcon[x]*, where $x = i/4$, the result being assigned to a 4-byte word *T* (935). *Rcon[x]* returns a 4-byte vector, *Rcon[x]* = (*RC(x)*, '00', '00', '00'), where the values of *RC[x]* are as follows:

RC[1] = '01'	RC[2] = '02'	RC[3] = '04'	RC[4] = '08'	RC[5] = '10'
RC[6] = '20'	RC[7] = '40'	RC[8] = '80'	RC[9] = '1B'	RC[10] = '36'

[0049] The word W[*i*-4] is then XORed with the word currently assigned to *T* to produce the next word W[*i*] (940).

[0050] The value of counter *i* is then tested to check if all the words of the expanded key have been produced (945). If *i* is not less than 43 then the expanded key is complete.

[0051] To perform round key selection, a second counter j (which represents a round key index) is set to zero (960). Four 4-byte words W[4j] to W[4j+3] are assigned to Round Key[j] (965) forj = 0 to 10 (965, 970), j being incremented in steps of 1 (975). Thus, for a ten round encryption/decryption, eleven round keys are provided, round key 0 to round key 10 where round key 0 comprises words W[0] to W[3] of the expanded key (i.e. the original cipher key), round key 1 comprises words W[4] to W[7] of the expanded key, and so on (See Fig. 1c). Round key 0 is used by the data/key addition module 48, round key 1 is provided to the round module 44 for round 1, round key 2 is provided to the round module 44 for round 2 and so on until round key 10 is used in the round module 46 for the final round (see Figs 4 and 5).

[0052] The round keys are created as required, hence, round key 0 is available immediately, round key 1 is created one clock cycle later and so on.

[0053] Figure 8 shows a flowchart illustrating the implementation of the Rijndael key schedule 28 for use in decryption. Key expansion is performed from the inverse cipher key so that the words 17 of the expanded key are produced in the order that they are required for decryption. Hence, in module 1005, the words 17 of the inverse cipher key are assigned to $W[(N_b*(N_r+1))-N_k]$ to $W[(N_b*(N_r+1))-1]$ respectively and, in module 1010, counter *i* is set to $(N_b*(N_r+1))-1)$ and decremented by 1 (module 1046) after each new word $W[i-N_k]$ is produced until $i = N_k$. The flowchart of Figure 8 shows the implementation of the key schedule for = 4, 6 or 8 and will be readily understood by a skilled person by analogy with Figures 6, 6a, 6b and 7.

[0054] There are a number of ways in which the flow charts of Figures 6, 6a, 6b, 7 and 8 can be implemented using, for example, direct hardware design or using conventional hardware description language (HDL), such as VHDL, together with conventional hardware synthesis tools.

As is now described, the present invention provides an apparatus for production of encryption/decryption keys. The apparatus of the invention is particularly suited for efficient implementation of key expansion in accordance with the Rijndael key schedule.

[0055] Figure 9 shows an apparatus 100 according to the invention for generating encryption keys and, in particular, for implementing Rijndael key expansion as shown in the flow chart of Figure 7. The apparatus 100 comprises a shift register 101, or similar data storage means, for storing the cipher key and sub-keys generated from the cipher key. In particular, the shift register 101 is arranged to store the cipher key initially and then to store each subsequent vector or word 17 of the expanded key as it is created. The arrangement is such that, as each newly created word 17 of the expanded key is input to the shift register 101, a word of the cipher key (and subsequently of the expanded key) is displaced and output from the shift register 101. The size of the shift register 101 is equal to the size of the cipher key length. For implementing the Rijndael key schedule, the size of the shift register is N_k *x 4 bytes*. Thus, when $N_k = 4$, the shift register 101 comprises four 4-byte registers, or storage locations, and so on.

[0056] The shift register 101 has an initialization input 103, by which data can be supplied to a first storage location 105, and an output 107, by which data can be displaced from a final storage location 109. Between the first and final storage locations 105, 109, the shift register 101 comprises N_k-2 intermediate storage locations 111. In the present embodiment, each storage location 105, 109, 111 is 4-bytes in size to accommodate the 4-byte words 16, 17 that make up the cipher key and the expanded key respectively. The shift register 101 has a second input 113 by which data can

be supplied to the first storage location 105.

[0057] The shift register 101 operates in normal manner - the respective contents of each register storage location are shifted through the shift register from one storage location to the next in successive operational cycles, the operational cycles typically being governed by a clock signal (not shown). Thus, when a block, in the present embodiment a 4-byte word, of data is supplied to an input 103, 113 of the shift register 101, it is placed in the first storage location 105. In the same clock cycle, the data block that had been stored in the final storage location 109 is displaced from the shift register 101 via output 107 and the data blocks stored in the intermediate storage locations 111 are shifted to the adjacent or successive storage location 111, 109 in the direction indicated by arrow A (i.e. towards the final storage location). In this way, a data block enters the shift register in the first storage location 105 and is shifted through the intermediate storage locations 111 consecutively as each subsequent data block enters the first storage location 105 until it reaches the final storage location 109 whereupon it is displaced from the shift register 101 via output 107 upon receipt of the next new data block in the first storage location 105. If the shift register 101 is empty to begin with, then each storage location may be loaded with a respective data block by inputting data blocks in sequence into the first storage location - as each successive data block is input, the preceding data block or blocks are shifted through the shift register 101 one storage location at a time until the shift register 101 is full.

[0058] A conventional shift register or other data buffer device, such as a FIFO (First-In First-Out) memory, is suitable for use as the shift register 101.

[0059] The apparatus 100 is generic and shows how to implement the Rijndael key schedule 28 when $N_k = 4$, 6 or 8. The apparatus includes circuitry 115 for performing appropriate transformations and logical operations on the data stored in the first storage location 105 and the data stored in the final storage location 109 to produce the next data block for storage in the first storage location 105. Initially, the cipher key W[0] to W[N_k-1] is loaded into the N_k storage locations of the shift register 101 via input 103 in conventional manner such that W[0] is held in the final storage location 109 and W[N_k-1] is held in the first storage location 105. The circuitry 115 is then enabled to operate on W[0] and W[N_k-1] to produce the next word 17 of the expanded key namely W[N_k]. W[N_k] is then placed in the first storage location 105 via input 113.

In the same cycle, W[0] is shifted out of the shift register 101 via output 107. Thus, at the end of the first operational cycle of the apparatus 100, the shift register contains words W[1] to W[N_k], with W[1] in the final storage location 109, W[N_k] in the first storage location 105 and the intermediate words W[2] to W[N_k-1] in consecutive order in the intermediate storage locations 111. In the next operational cycle of the apparatus 100, the circuitry 115 performs the necessary transformations an other operations on words W[1] and W[N_k] to produce the next word 17 of the expanded key, namely W[N_k+1], which is then loaded into the first storage location 105 of the shift register 101 while W[1] is shifted out of the shift register 101. Thus, in each successive operational cycle of the apparatus 100, a new word 17 of the expanded key is created and the word 17 N_k positions in advance of the new word is output from the apparatus 100. The operation of the apparatus 100 continues in this way until the last word 17 of the expanded key, namely W[(N_b*(N_r+1))-1], is created. At this time, the shift register 101 contains the expanded key words W[(N_b*(N_r+1))-N_k] to [(N_b*(N_r+1))-1]. The circuitry 115 is then disabled and the expanded key words remaining in the shift register 101 are shifted out of the register 101 in conventional manner.

[0060] The circuitry 115 is arranged to perform the Rijndael transformations and other operations as described above and illustrated in the flow chart of Figure 7. The circuitry 115 includes a RotByte module 117 for performing a cyclic shift to the left of each byte in the 4-byte word. This may conveniently be implemented by hardwiring. The circuitry also includes a SubByte module 119 for performing the Rijndael ByteSub transformation. Conveniently, the SubByte module 119 comprises one or more Look-Up Tables (LUT) (not shown). Each byte of each word 17 passed through the SubByte module 119 is input to a LUT to produce a corresponding 8-bit output. Figure 11 shows two tables of values suitable for use in a LUT for implementing the Rijndael ByteSub transformation. For example, if the input byte 'B3' (hexadecimal) is input to a LUT containing these values, then the 8-bit output returned by the LUT is '6D', while if the input byte is '5A', the output byte is 'BE', and so on. LUTs can be implemented in a number of conventional ways using, for example, RAMs or ROMs.

[0061] The circuitry 115 also includes a *Rcon* module 121 for implementing the *Rcon*(x) function described above, where x = i/N_k, i representing a counter that counts the operation cycles of the apparatus 100 and corresponds with an index to the words 17 of the expanded key.

[0062] Counter i starts at N_k and increments by 1 for each operational cycle of the apparatus 100 up to [(N_b*(N_r+1))-1]. For $i = 0$ to N_k-1, the circuitry 115 is disabled and the cipher key is loaded into the shift register 101. For $i = N_k$ to [(N_b*(N_r+1))-1], the circuitry is enabled and the words of the expanded key are generated as described above.

[0063] The *Rcon* module 121 may conveniently be implemented by means of a LUT. The respective outputs of the *Rcon* module 121 and the SubByte module 119 are XORed by gate 123.

[0064] In order to implement the variations required by Rijndael, the circuitry 115 includes a switching mechanism 125 whereby one or other of terminals T1, T2 and T3 may be selected at one time. The selection position adopted by the switch 125 is controlled by the value of counter i. Normally, the switch 125 selects terminal T1. In this state, the

respective words in the first and final register storage locations 105, 109 are XORed by gate 127 to produce the next word 17 of the expanded key. When *i rem* $N_k = 0$, the switch 125 selects terminal T2 whereupon the word stored in the first storage location 105 is passed through the RotByte module 117, SubByte module 119 and XOR gate 123 before being XORed with the contents of the final location 109 by gate 127. When $N_k = 8$ and i *rem* 8 = 4, the switch 125 selects terminal T3 whereupon word stored in the first storage location 105 is passed through a SubByte module 119' before being XORed with the contents of the final location 109 by gate 127.

[0065] The counter *i* may be implemented in any convenient conventional manner and used, as described above, to in the calculation of the *Rcon* and *rem* functions. The *rem* function may be implemented in any convenient manner, for example by a LUT (not shown) or by a conventional comparator module (not shown) arranged to compare the values of *i* with known multiples of N_k.

[0066] The shift register 101 shifts data every clock cycle. In order to synchronize the operation of the apparatus 100, i.e. to synchronize the flow of data words in the apparatus 100, a further data register (not shown) is included in the apparatus 100. Conveniently, the further data register is included in the SubByte module 119 since, in the preferred embodiment, the SubByte module 119 is implemented by one or more LUTs, which typically comprise a RAM(s) or ROM(s) which, in turn, typically include a data register in their architecture. The shift register 101 and the further register are synchronized to a common clock signal in conventional manner. The encryption or decryption apparatus of which the apparatus of the invention is part, is also synchronized to the common clock signal.

[0067] Figure 9a shows, by way of example, a schematic view of an apparatus 100' for implementing the Rijndael key expansion where $N_k = 4$ (corresponding to the flow chart of Figure 6). In this embodiment, it will be seen that the switch 125' need only select either terminal T1 or T2 (T2 is selected when *i* rem 4 = 0). The shift register 101' is a 4-word shift register (which in this case is a 4 x 4-byte shift register). Initially, the shift register 101' is loaded with the cipher key W[0] to W[3] in four cycles where *i* = 0 to 3. In the cycle where *i* = 4, W[0] is shifted out of the register 101' via output 107' and a new word W[4] is created by the circuitry 115' and stored in the first storage location 105. Hence, the shift register 101' now contains W[1] (in the final location 109'), W[2], W[3] (in the intermediate locations 111') and W[4]. The process repeats for *i* = 5 to 43. When *i* = 43, the shift register 101' contains W[40] (in the final location 109'), W[41], W[42] (in the intermediate locations 111') and W[43] in the first location 105. These words 17 can then be read from the shift register 101' in normal manner.

[0068] Figure 9b shows a further embodiment of the invention in which the apparatus 100" is able to support either a 128-bit, 192-bit or 256-bit cipher key depending on the setting of first and second switches 143, 145. The apparatus 100" comprises a shift register 101" having eight storage locations 111". The switches 143, 145 each have three selectable terminals S1, S2, S3 which connect the circuitry 115" with respective storage locations of the shift register 101". The setting of the switches 143, 145 determines the effective size of the shift register 101" and also determines which of the storage locations 111" serves as said first storage location 105". The shift register 101" is loaded initially with the N_k-word cipher key in conventional manner. When $N_k = 4$, the switches 143, 145 are arranged to select terminals S1 and so only four storage locations 111" of the shift register 101" are used. When $N_k = 6$, the switches 143, 145 are arranged to select terminals S2 and only six storage locations of the shift register 101" are used. When $N_k = 8$, the switches are arranged to select terminals S3 and all eight storage locations of the shift register 101" are used.

[0069] Figure 10 illustrates a schematic view of a further embodiment of the invention in the form of an apparatus 200 for implementing the Rijndael key schedule 28 for data decryption. The apparatus 201 implements the key expansions operations illustrated in Figure 8. The apparatus 200 is generally similar in structure to the apparatus 100 and includes a shift register 201 and circuitry 215 for performing the required Rijndael transformations and other operations. To this end, the apparatus 200 includes a Rotbyte module 217, SubByte modules 219, an *Rcon* module 221, XOR gates 223, 227 and a switching mechanism 125 in similar arrangement to the apparatus 100. However, in the apparatus 200, the circuitry 215 operates on the data, i.e. words of the inverse cipher key and the expanded key, contained in the final storage location 209 of the shift register 201 and the penultimate storage location 211a of the shift register 201. Initially, the shift register 201 is loaded with the inverse cipher key $W[(N_b*(N_r+1))-N_k]$ to $W[(N_b*(N_r+1))-1]$ in consecutive order such $W[(N_b*(N_r+1))-1]$ is stored in the final storage location 209 and $W[(N_b*(N_r+1))-N_k]$ is stored in the first storage location 205. The apparatus 200 operates in substantially similar manner to the apparatus 100. However, counter *i* is initialized to the value $N_b*(N_r+1)-1$ and is decremented by 1 for each operational cycle of the apparatus 200 until $i = N_k$.

[0070] It will be seen that the apparatus 200 produces the words 17 of the expanded key in the order required for decryption, i.e. reverse order, each successive word being shifted out of the shift register 201 in consecutive operation cycles of the apparatus 200.

[0071] Figure 10a illustrates, by way of example, a schematic view of an apparatus 200' for implementing the Rijndael key expansion as shown in the flow chart of Figure 8 for where $N_k = 4$. As for the apparatus 100', it will be seen that the switch 225' need only select either terminal T1 or T2 (T2 is selected when *i rem* 4 = 0). The shift register 201' is a 4 x 4-byte shift register. Initially, the shift register 201' is loaded with the inverse cipher key W[43] to W[40]. In the subsequent cycle, W[43] is shifted out of the register 201' via output 207' and a new word W[39] is created by the

circuitry 215' and stored in the first storage location 205. Hence, the shift register 201' now contains W[42] (in the final location 209'), W[41], W[40] (in the intermediate locations 211') and W[39]. The process repeats until the shift register 201' contains W[3] (in the final location 209'), W[2], W[1] (in the intermediate locations 211') and W[0] in the first location 205. These words 17 can then be read from the shift register 201' in normal manner.

[0072] Figure 10b shows a further embodiment of the invention in which the apparatus 200" is able to support either a 128-bit, 192-bit or 256-bit cipher key depending on the setting of a switch 243. The apparatus 200" comprises a shift register 201" having eight storage locations 211". The switch 243 has three selectable terminals S1, S2, S3 which connect the circuitry 215" with respective storage locations of the shift register 201". The setting of the switch 243 determines the effective size of the shift register 201" and also determines which of the storage locations 211" serves as said first storage location 205". The shift register 201" is loaded initially with the N_k-word cipher key in conventional manner. When $N_k = 4$, the switch 243 is arranged to select terminal S1 and so only four storage locations of the shift register 201" are used. When $N_k = 6$, the switch 243 is arranged to select terminal S2 and only six storage locations of the shift register 201" are used. When $N_k = 8$, the switch is arranged to select terminal S3 and all eight storage locations of the shift register 201" are used.

[0073] In Figures 9, 9a, 10, 10a, the shift registers 101, 101', 201, 201' are shown with two inputs to the first storage location 105, 105', 205, 205' for clarity. In practice, a single input may be provided for performing all input operations to the shift registers 101, 101', 201, 201'.

[0074] It will be understood from the foregoing that, after an initial delay of N_k clock cycles to allow the cipher key/inverse cipher key to be loaded into the shift register 101, 101', 101", 201, 201', 201", the expanded key is output from the apparatus 100, 100', 100", 200, 200', 200" one word 17 at a time and in successive clock cycles. Moreover, by initializing the shift register 101, 101', 101", 201, 201', 201" with the cipher key or inverse cipher key as appropriate, the words are produced in the order that they are required by the surrounding encryption apparatus or decryption apparatus. The apparatus of the invention is particularly suited for use with an encryption/decryption apparatus in which each encryption or decryption round is performed over a plurality of successive clock cycles using the same round module. By way of example the apparatus 100, 100', 100" are suitable for use as the key scheduler 50 of the encryption apparatus 40 of Figure 5a, while the apparatus 200, 200', 200" are suitable for use as the key scheduler 50' of the decryption apparatus 40' of Figure 5b.

[0075] The embodiments described herein relate primarily to the case where the data block length, N_b, is 128-bits, the round is performed over four clock cycles and the key scheduling apparatus 100, 100', 100", 200, 200', 200" have a 4-register shift register, thus producing a round key every four cycles. In the case of a 192-bit data block, the round will be performed over 6 clock cycles, the key scheduling apparatus has a 6-register shift register and produces a round key every six clock cycles. For a 256-bit data block the round is performed over 8 clock cycles and the corresponding key scheduling apparatus has a 6-register shift register and creates a round key every 8 clock cycles.

[0076] It will be noted that the apparatus 200, 200', 200" are arranged to perform, in particular, on-the-fly Rijndael decryption Round key calculation. This is particularly advantageous as it obviates the need to store the expanded key or to wait until the expanded key is generated from the cipher key before beginning decryption. This removes a latency of at least 10 clock cycles in the operation of the decryption apparatus. Further, the use of the shift register 101, 101', 101", 201, 201', 201" in the manner described above results in the apparatus of the invention being smaller, in terms of gate count and physical size, than conventional implementations which may use, for example, RAMs and multiplexers.

[0077] The apparatus 100, 100', 100", 200, 200', 200" may be implemented on an FPGA device or other conventional devices such as other Programmable Logic Devices (PLDs) or an ASIC (Application Specific Integrated Circuit). In an ASIC implementation, the LUTs may be implemented in conventional manner using, for example, standard RAM or ROM components.

[0078] The invention is not limited to the embodiments described herein which may be modified or varied without departing from the scope of the invention.

Claims

1. An apparatus for generating a plurality of sub-keys from a primary key comprising a plurality of data words, the apparatus comprising: a shift register having a plurality of storage locations one for each data word of the primary key; and a transformation apparatus arranged to perform one or more logical operations on respective data words from at least two of said storage locations to produce a new data word, the arrangement being such that said new data word is loaded into a first of said storage locations, whereupon the data words stored in said shift register are shifted to a respective successive storage location and the data word in a final of said storage locations is output from said shift register, said sub-keys being comprised of one or more of said output data words.

2. An apparatus as claimed in Claim 1, wherein said new data word is loaded into said first storage location via a first switch, said switch being arranged to select which of said storage locations serves as said first storage location.

3. An apparatus as claimed in Claim 1 or 2, wherein at least one data word is provided to said transformation module from said shift register via a second switch, the second switch being arranged to select from which storage location said at least one data word is provided.

4. An apparatus as claimed in any preceding claim, wherein the transformation apparatus is arranged to perform transformations according to the Rijndael block cipher.

5. An apparatus as claimed in Claim 4, wherein the shift register is initialised with a primary key comprising a Rijndael cipher key and said transformation apparatus is arranged to perform said one or more logical operations on the respective data words stored in said first and said final storage locations.

6. An apparatus as claimed in Claim 4, wherein the shift register is initialised with a primary key comprising a Rijndael inverse cipher key and said transformation apparatus is arranged to perform said one or more logical operations on the respective data words stored in said final storage location and the penultimate storage location.

7. A method of generating a plurality of sub-keys from a primary key comprising a plurality of data words, method comprising:

loading the primary key into a shift register having a plurality of storage locations one for each data word of the primary key;

performing one or more logical operations on respective data words from at least two of said storage locations to produce a new data word;

loading said new data word into a first of said storage locations,

whereupon the data words stored in said shift register are shifted to a respective successive storage location and the data word in a final of said storage locations is output from said shift register, said sub-keys being comprised of one or more of said output data words.

8. A data encryption and/or decryption apparatus comprising an apparatus for generating a plurality of sub-keys as claimed in Claim 1.

9. A computer program product comprising computer usable instructions for generating an apparatus according to Claim 1.

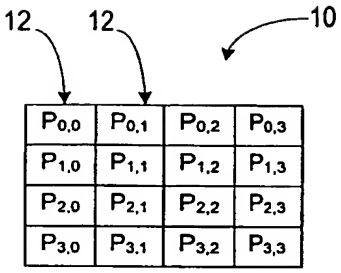

Fig. 1a

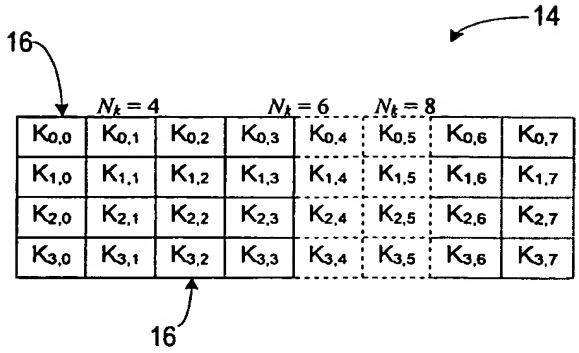

Fig. 1b

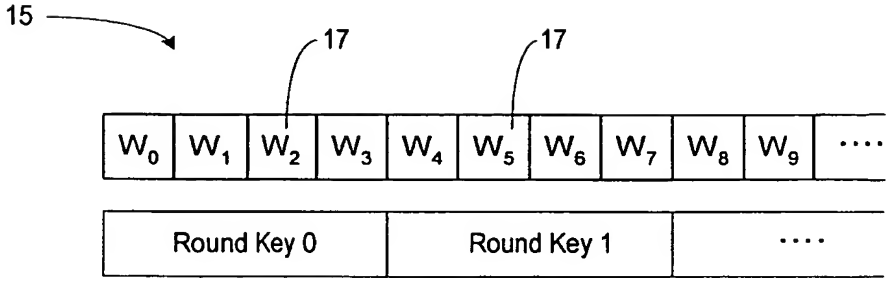

Fig. 1c

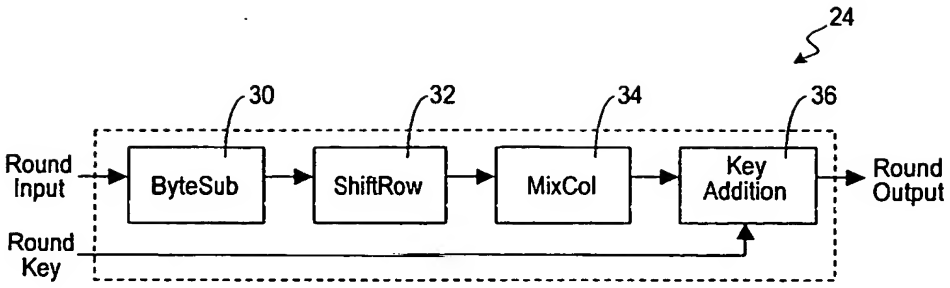

Fig. 3

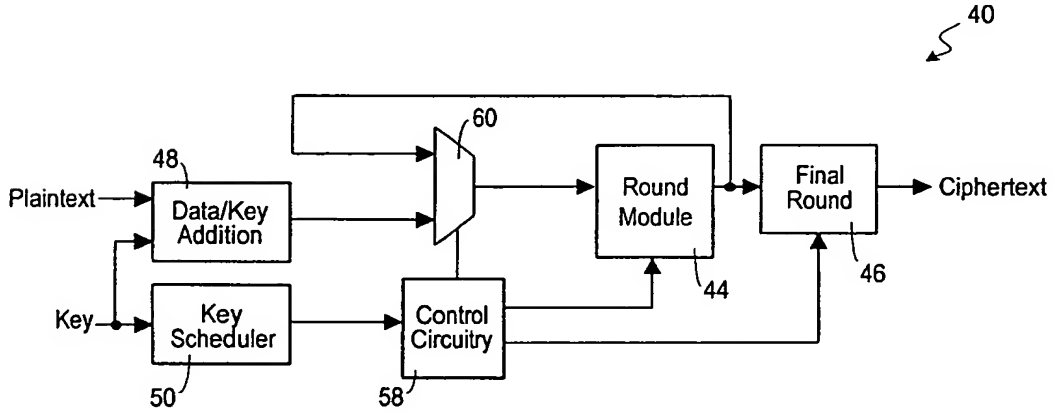

Fig. 5a

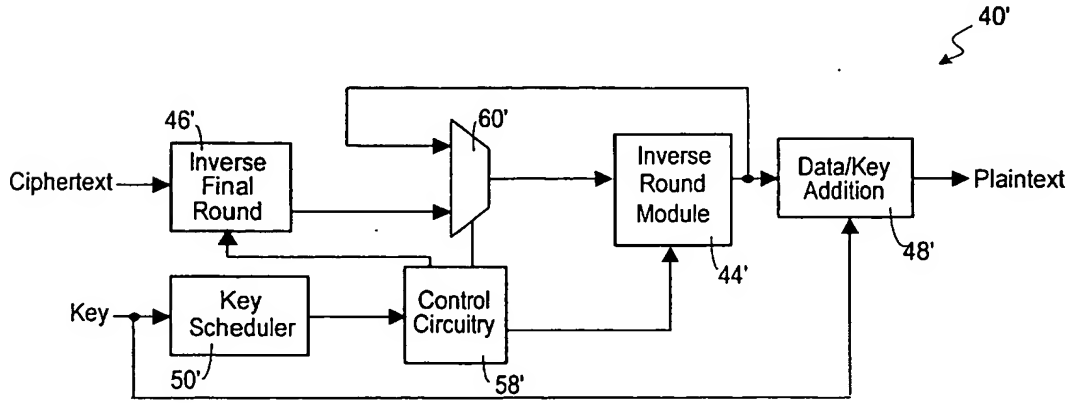

Fig. 5b

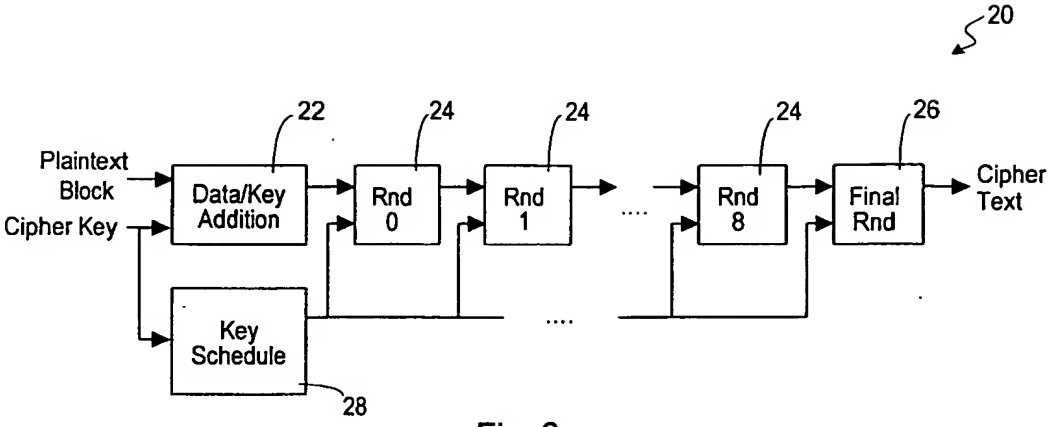

Fig. 2

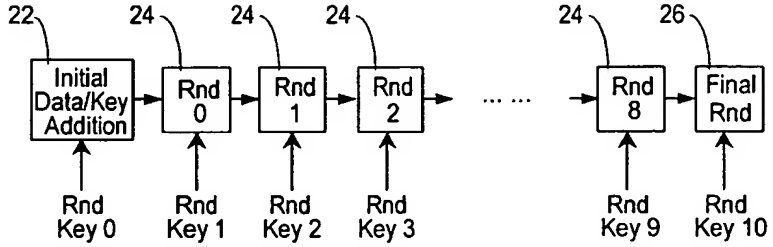

Fig. 4

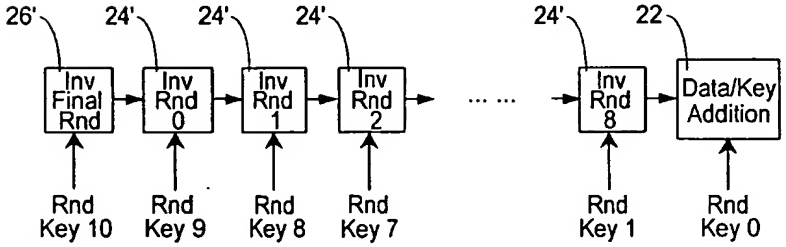

Fig. 4a

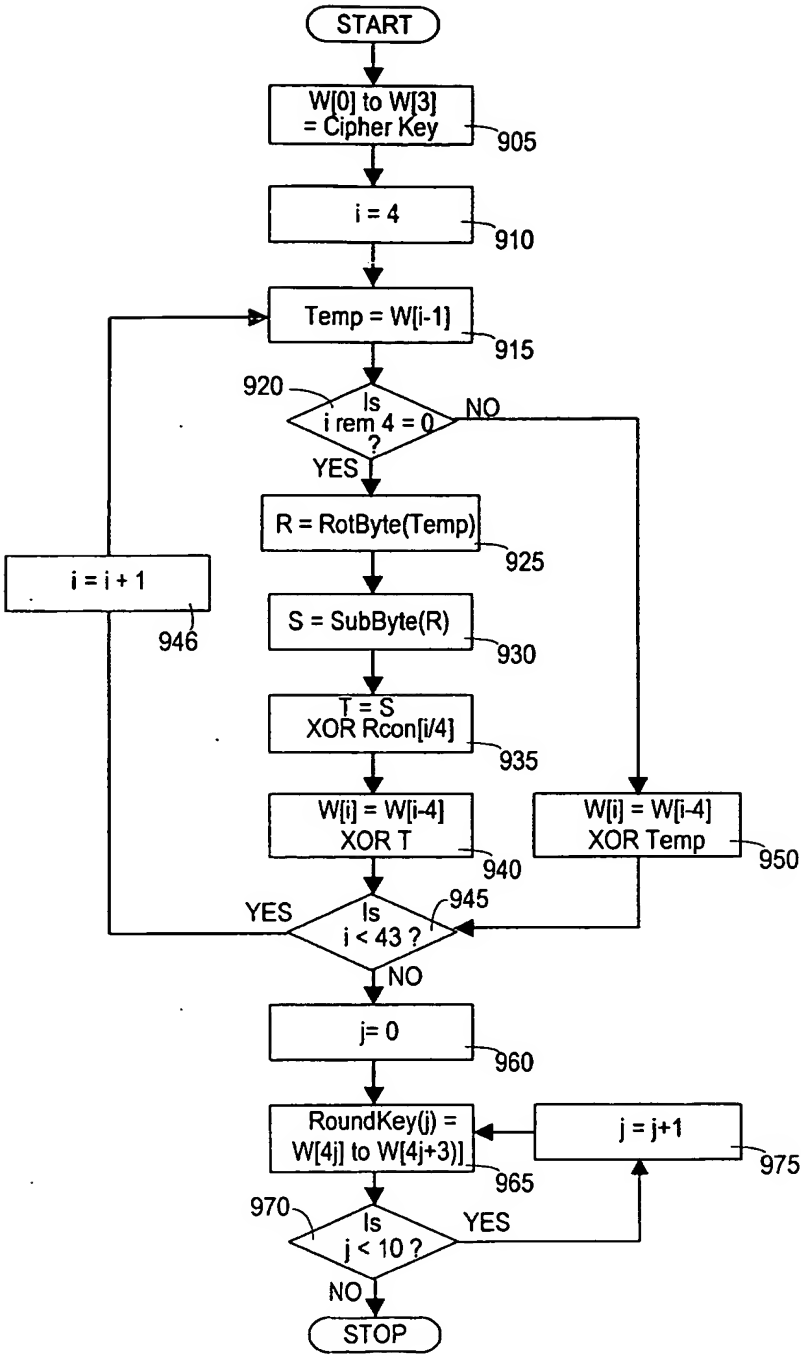
START
W[0] to W[3] = Cipher Key
905
i = 4
910
Temp = W[i-1]
915
920
Is i rem 4 = 0 ?
NO
YES
R = RotByte(Temp)
925
i = i + 1
946
S = SubByte(R)
930
T = S XOR Rcon[i/4]
935
W[i] = W[i-4] XOR T
940
W[i] = W[i-4] XOR Temp
950
945
YES
Is i < 43 ?
NO
j= 0
960
RoundKey(j) = W[4j] to W[4j+3)]
965
j = j+1
975
970
Is j < 10 ?
YES
NO
STOP

Fig. 6

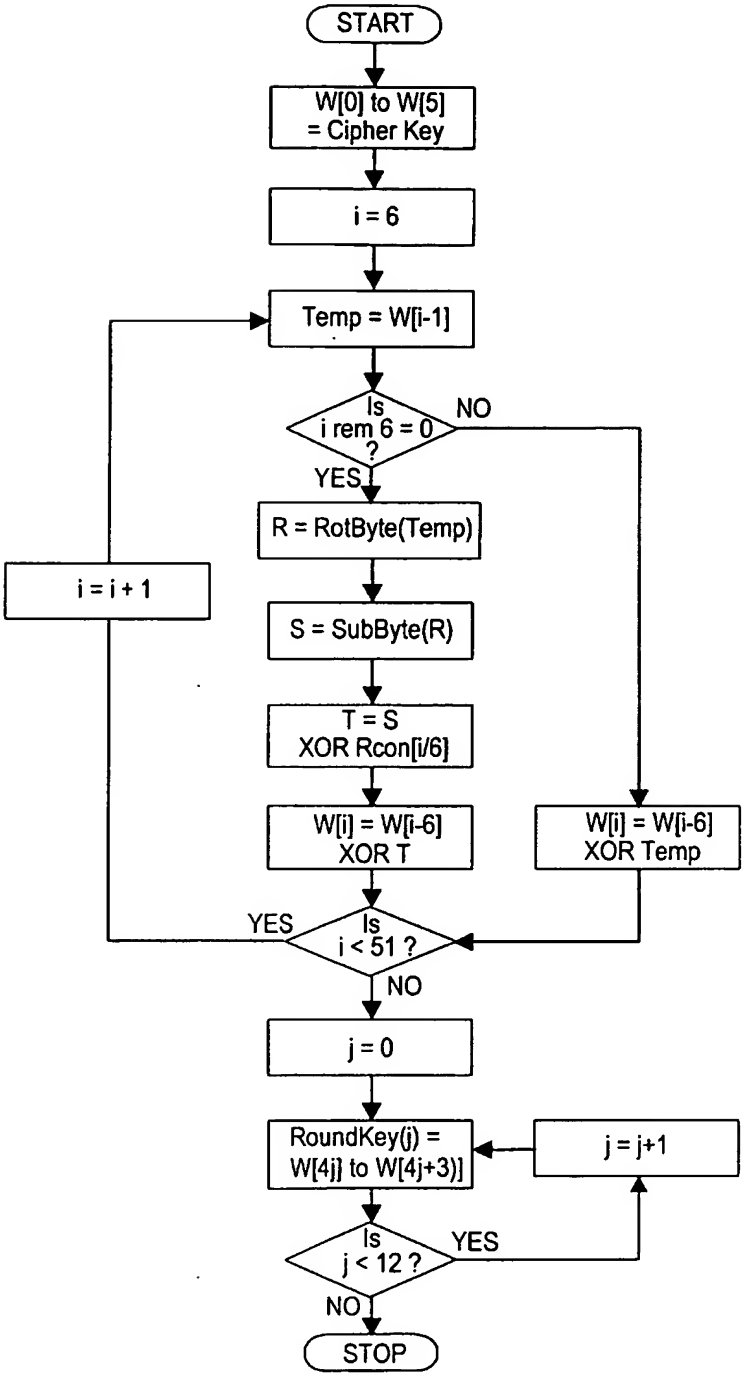
START
W[0] to W[5] = Cipher Key
i = 6
Temp = W[i-1]
Is i rem 6 = 0 ?
NO
YES
R = RotByte(Temp)
S = SubByte(R)
T = S XOR Rcon[i/6]
W[i] = W[i-6] XOR T
W[i] = W[i-6] XOR Temp
i = i + 1
YES
Is i < 51 ?
NO
j = 0
RoundKey(j) = W[4j] to W[4j+3)]
j = j+1
Is j < 12 ?
YES
NO
STOP

Fig. 6a

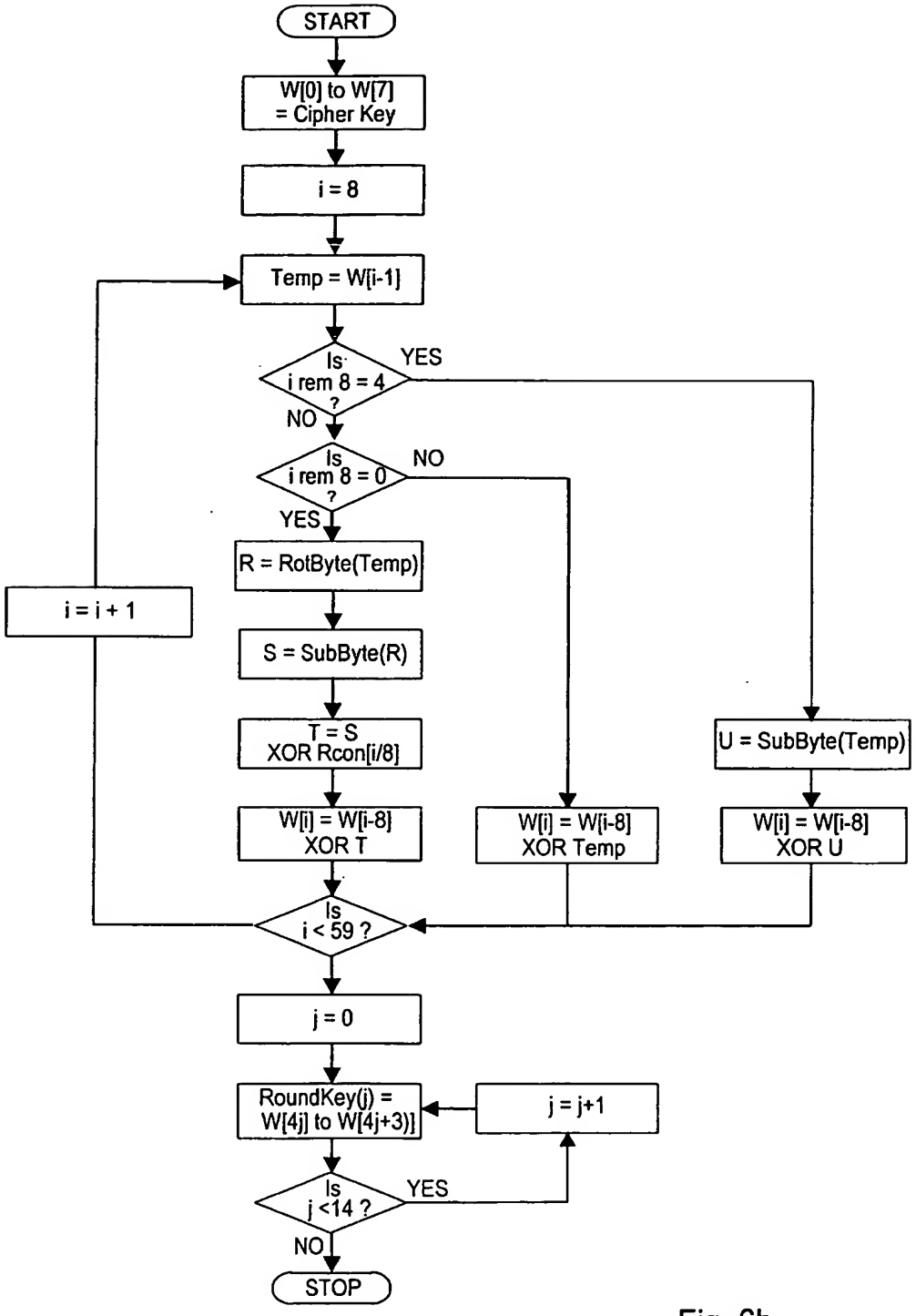
START
W[0] to W[7] = Cipher Key
i = 8
Temp = W[i-1]
Is i rem 8 = 4 ?
YES
NO
Is i rem 8 = 0 ?
NO
YES
R = RotByte(Temp)
S = SubByte(R)
T = S XOR Rcon[i/8]
W[i] = W[i-8] XOR T
W[i] = W[i-8] XOR Temp
U = SubByte(Temp)
W[i] = W[i-8] XOR U
i = i + 1
Is i < 59 ?
j = 0
RoundKey(j) = W[4j] to W[4j+3)]
j = j+1
Is j <14 ?
YES
NO
STOP

Fig. 6b

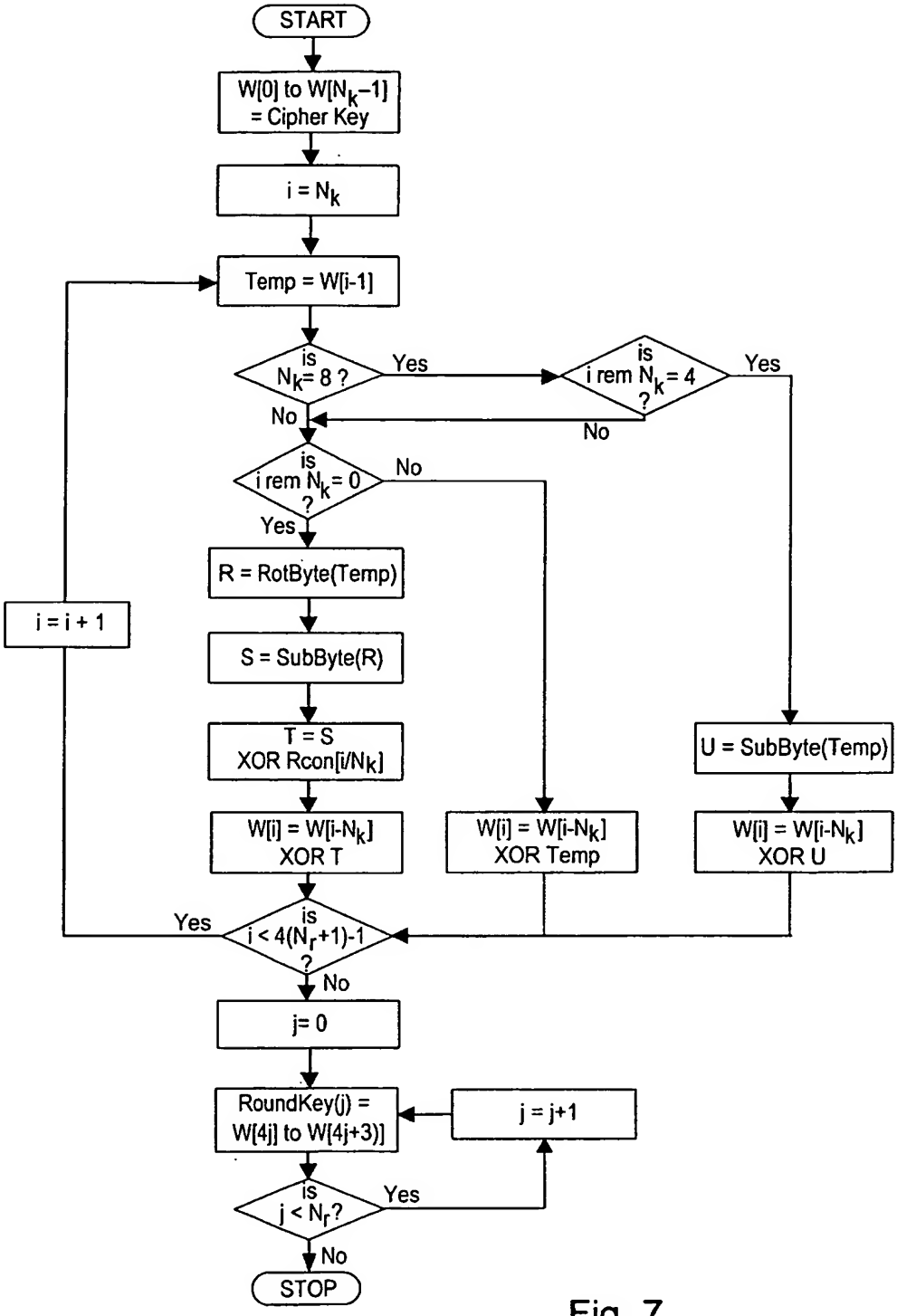
START
W[0] to W[Nk−1] = Cipher Key
i = Nk
Temp = W[i-1]
is Nk= 8 ?
Yes
is i rem Nk= 4 ?
Yes
No
No
is i rem Nk= 0 ?
No
Yes
R = RotByte(Temp)
S = SubByte(R)
T = S XOR Rcon[i/Nk]
W[i] = W[i-Nk] XOR T
W[i] = W[i-Nk] XOR Temp
U = SubByte(Temp)
W[i] = W[i-Nk] XOR U
i = i + 1
Yes
is i < 4(Nr+1)-1 ?
No
j= 0
RoundKey(j) = W[4j] to W[4j+3)]
j = j+1
is j < Nr?
Yes
No
STOP

Fig. 7

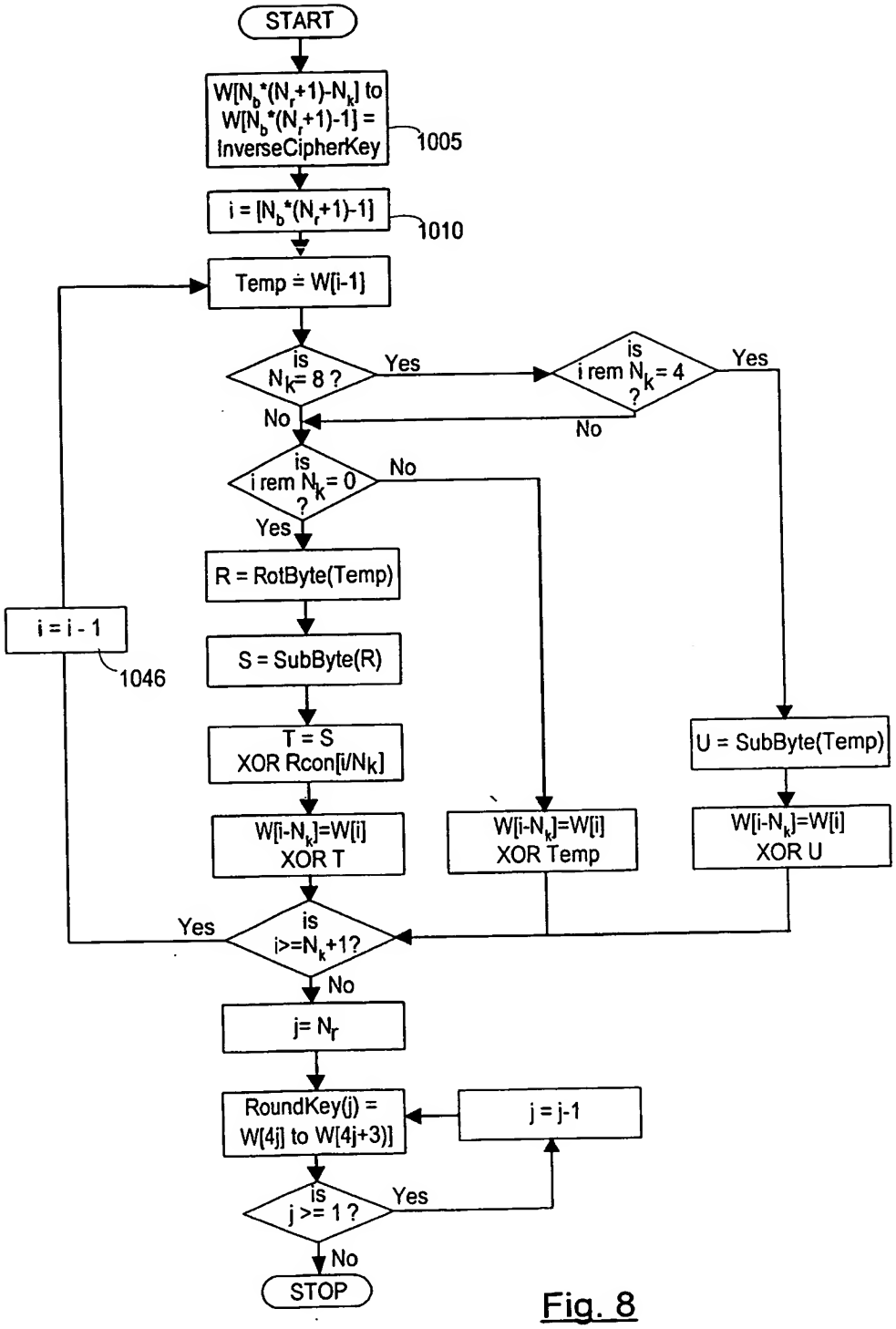
START
W[Nb*(Nr+1)-Nk] to W[Nb*(Nr+1)-1] = InverseCipherKey
1005
i = [Nb*(Nr+1)-1]
1010
Temp = W[i-1]
is Nk= 8 ?
Yes
No
is i rem Nk= 4 ?
Yes
No
is i rem Nk= 0 ?
No
Yes
R = RotByte(Temp)
S = SubByte(R)
T = S XOR Rcon[i/Nk]
W[i-Nk]=W[i] XOR T
i = i - 1
1046
W[i-Nk]=W[i] XOR Temp
U = SubByte(Temp)
W[i-Nk]=W[i] XOR U
is j>=Nk+1?
Yes
No
j= Nr
RoundKey(j) = W[4j] to W[4j+3)]
j = j-1
is j >= 1 ?
Yes
No
STOP

Fig. 8

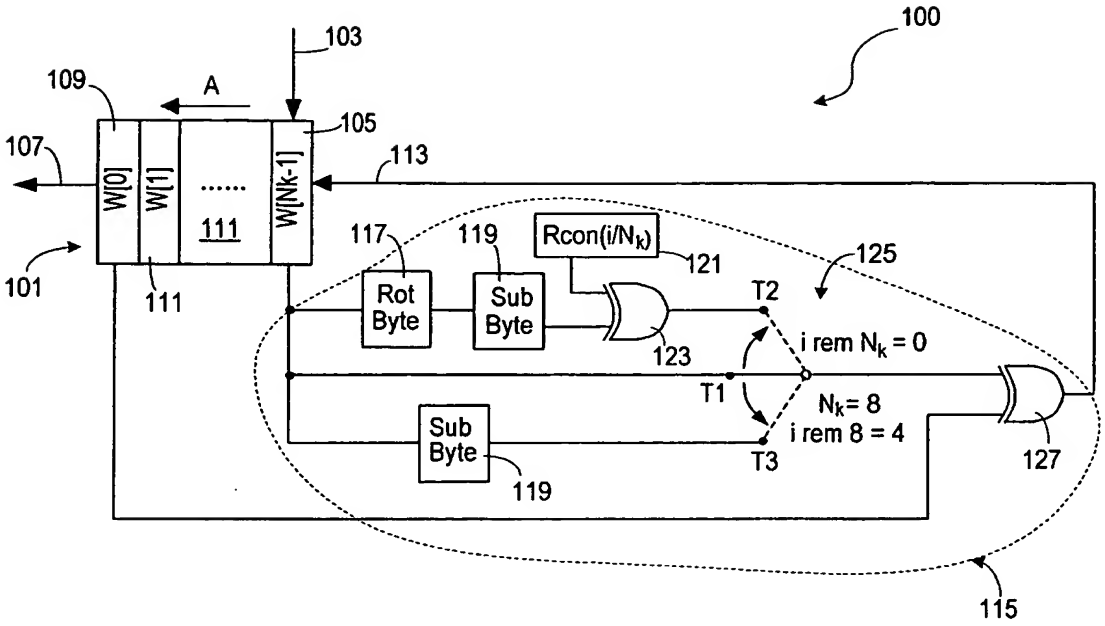
100
103
109
A
105
107
113
W[0]
W[1]
.....
W[Nk-1]
111
101
111
117
119
Rcon(i/Nk)
121
125
Rot Byte
Sub Byte
T2
123
i rem Nk = 0
T1
Nk= 8
i rem 8 = 4
Sub Byte
T3
127
119
115

Fig. 9

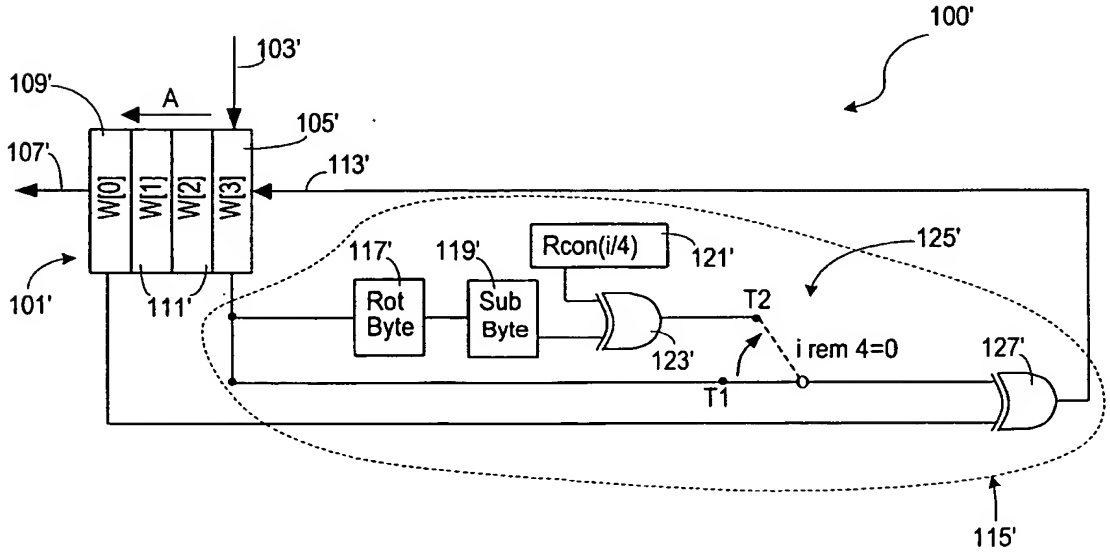
100'
103'
109'
A
105'
107'
113'
W[0]
W[1]
W[2]
W[3]
101'
111'
117'
119'
Rcon(i/4)
121'
125'
Rot Byte
Sub Byte
T2
123'
i rem 4=0
127'
T1
115'

Fig. 9a

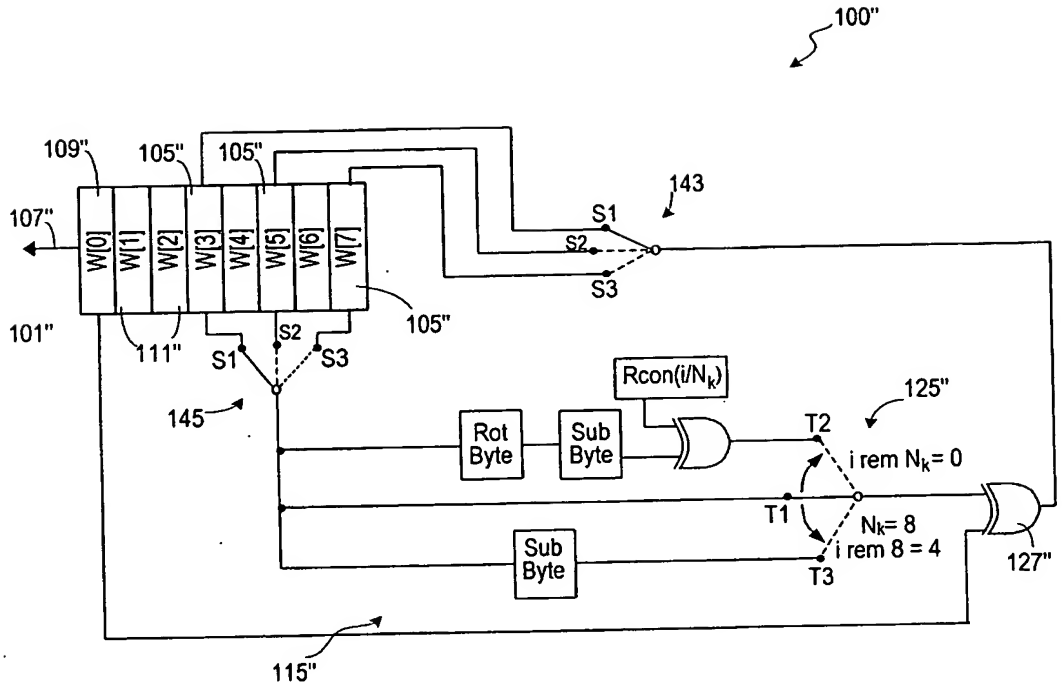
100"
109"
105"
105"
107"
W[0]
W[1]
W[2]
W[3]
W[4]
W[5]
W[6]
W[7]
143
S1
S2
S3
105"
101"
111"
S1
S2
S3
145
Rcon(i/N_k)
125"
Rot Byte
Sub Byte
T2
i rem N_k= 0
T1
N_k= 8
i rem 8 = 4
Sub Byte
T3
127"
115"

Fig. 9b

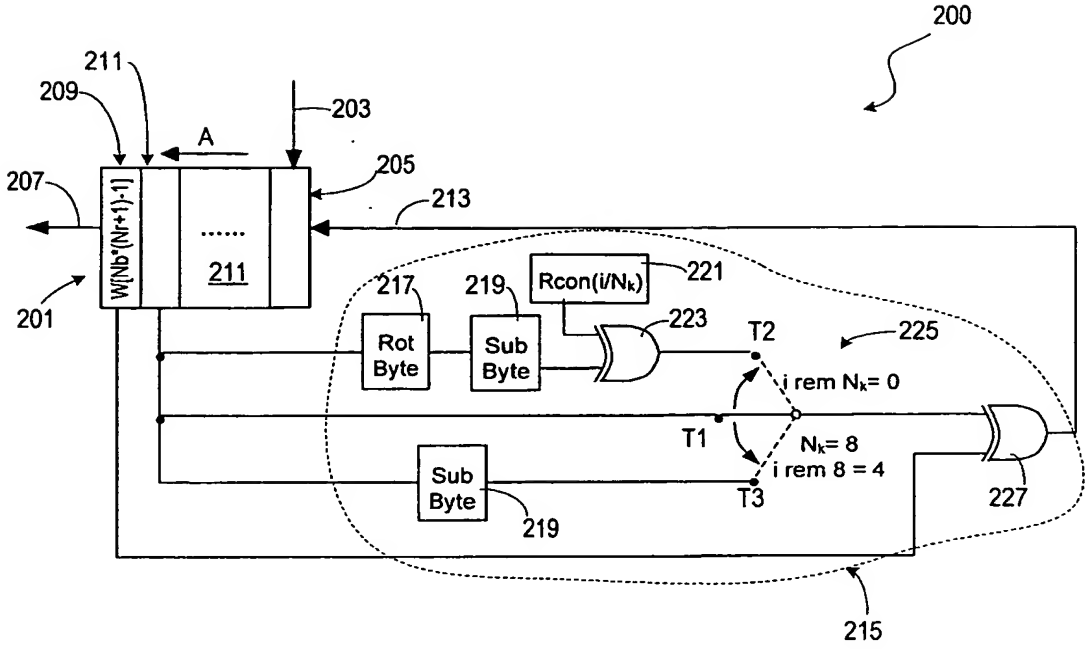

Fig. 10

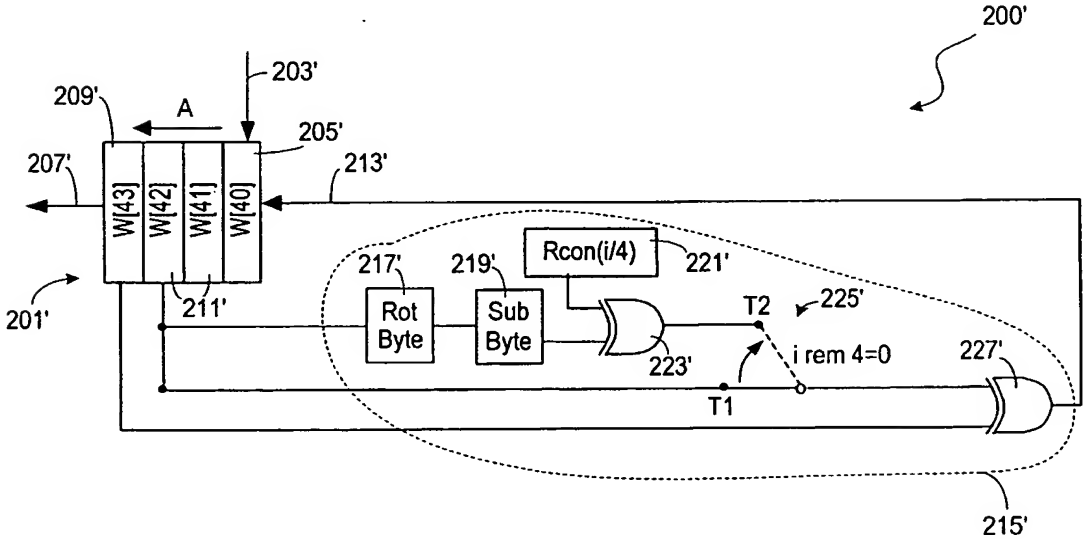

Fig. 10a

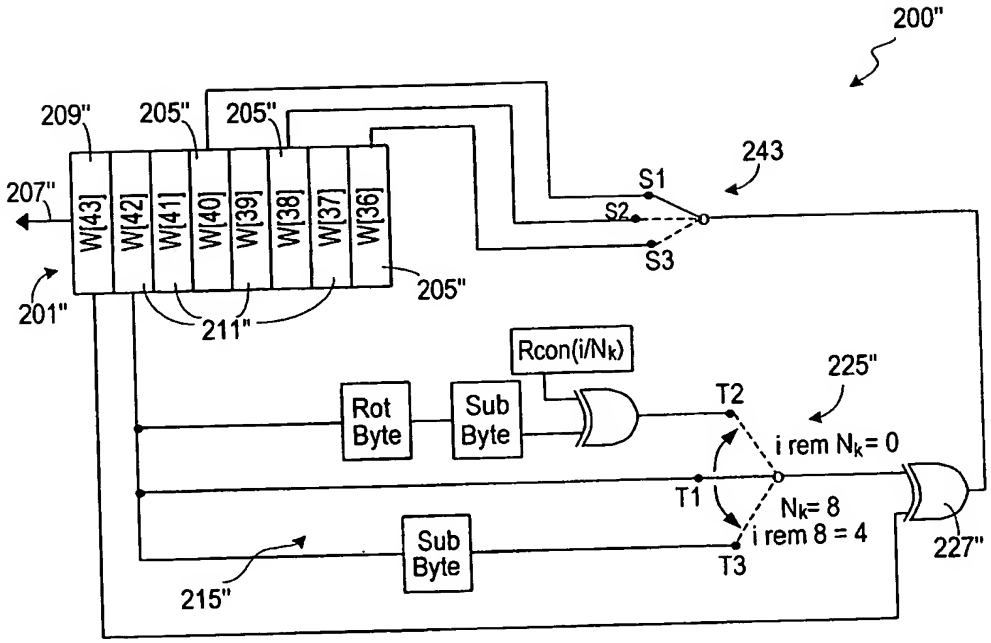
200"
209"
205"
205"
243
S1
S2
S3
207"
W[43]
W[42]
W[41]
W[40]
W[39]
W[38]
W[37]
W[36]
205"
201"
211"
Rcon(i/Nk)
225"
Rot Byte
Sub Byte
T2
i rem Nk= 0
T1
Nk= 8
i rem 8 = 4
Sub Byte
T3
227"
215"

Fig. 10b

	0	1	2	3	4	5	6	7
0	63	7C	77	7B	F2	6B	6F	C5
1	CA	82	C9	7D	FA	59	47	F0
2	B7	FD	93	26	36	3F	F7	CC
3	04	C7	23	C3	18	96	05	9A
4	09	83	2C	1A	1B	6E	5A	A0
5	53	D1	00	ED	20	FC	B1	5B
6	D0	EF	AA	FB	43	4D	33	85
7	51	A3	40	8F	92	9D	38	F5
8	CD	0C	13	EC	5F	97	44	17
9	60	81	4F	DC	22	2A	90	88
A	E0	32	3A	0A	49	06	24	5C
B	E7	C8	37	6D	8D	D5	4E	A9
C	BA	78	25	2E	1C	A6	B4	C6
D	70	3E	B5	66	48	03	F6	0E
E	E1	F8	98	11	69	D9	8E	94
F	8C	A1	89	0D	BF	E6	42	68

	8	9	A	B	C	D	E	F
0	30	01	67	2B	FE	D7	AB	76
1	AD	D4	A2	AF	9C	A4	72	C0
2	34	A5	E5	F1	71	D8	31	15
3	07	12	80	E2	EB	27	B2	75
4	52	3B	D6	B3	29	E3	2F	84
5	6A	CB	BE	39	4A	4C	58	CF
6	45	F9	02	7F	50	3C	9F	A8
7	BC	B6	DA	21	10	FF	F3	D2
8	C4	A7	7E	3D	64	5D	19	73
9	46	EE	B8	14	DE	5E	0B	DB
A	C2	D3	AC	62	91	95	E4	79
B	6C	56	F4	EA	65	7A	AE	08
C	E8	DD	74	1F	4B	BD	8B	8A
D	61	35	57	B9	86	C1	1D	9E
E	9B	1E	87	E9	CE	55	28	DF
F	41	99	2D	0F	B0	54	BB	16

Fig. 11

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 02 01 9688

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
X Y	EP 0 342 832 A (GEN INSTRUMENT CORP) 23 November 1989 (1989-11-23) * column 1, line 1 - line 20 * * column 1, line 53 - column 5, line 12 * * figures 1-4 * ---	1-3,7-9 4-6	H04L9/06
Y	DAEMEN J ET AL: "AES PROPOSAL: RIJNDAEL" AES PROPOSAL, XX, XX, 3 September 1999 (1999-09-03), pages 1-45, XP001060386 * page 8 - page 22 * ---	4-6	
Y	EP 0 750 399 A (MOTOROLA LTD) 27 December 1996 (1996-12-27) * abstract * * page 2, line 55 - page 4, line 5 * * figures 1,3 * ---	1	
Y	LEE S H ET AL: "Integrated parallel scrambler design for high-speed transmission systems" BELL COMMUNICATIONS RESEARCH, XP010069563 * the whole document * ---	1	TECHNICAL FIELDS SEARCHED (Int.Cl.7) H04L
A	MCLOONE M ET AL: "High performance single-chip FPGA Rijndael algorithm implementations" CRYPTOGRAPHIC HARDWARE AND EMBEDDED SYSTEMS. 3RD INTERNATIONAL WORKSHOP, CHES 2001, PARIS, FRANCCE, MAY 14 - 16, 2001 PROCEEDINGS, LECTURE NOTES IN COMPUTER SCIENCE, BERLIN: SPRINGER, DE, vol. 2162, 14 May 2001 (2001-05-14), pages 65-76, XP002203795 ISBN: 3-540-42521-7 * the whole document * -----	1-9	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
MUNICH	20 November 2002	Bec, T

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 02 01 9688

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

20-11-2002

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0342832 A	23-11-1989	US 4860353 A	22-08-1989
		AT 104100 T	15-04-1994
		AU 608439 B2	28-03-1991
		AU 3485989 A	23-11-1989
		CA 1331789 A1	30-08-1994
		DE 68914322 D1	11-05-1994
		DE 68914322 T2	23-11-1995
		DK 237189 A	18-11-1989
		EP 0342832 A2	23-11-1989
		ES 2097742 T3	16-04-1997
		IE 62615 B1	22-02-1995
		JP 2022693 A	25-01-1990
		JP 3024702 B2	21-03-2000
		NO 891934 A	20-11-1989
EP 0750399 A	27-12-1996	GB 2302634 A	22-01-1997
		US 5781566 A	14-07-1998
		EP 0750399 A2	27-12-1996

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82